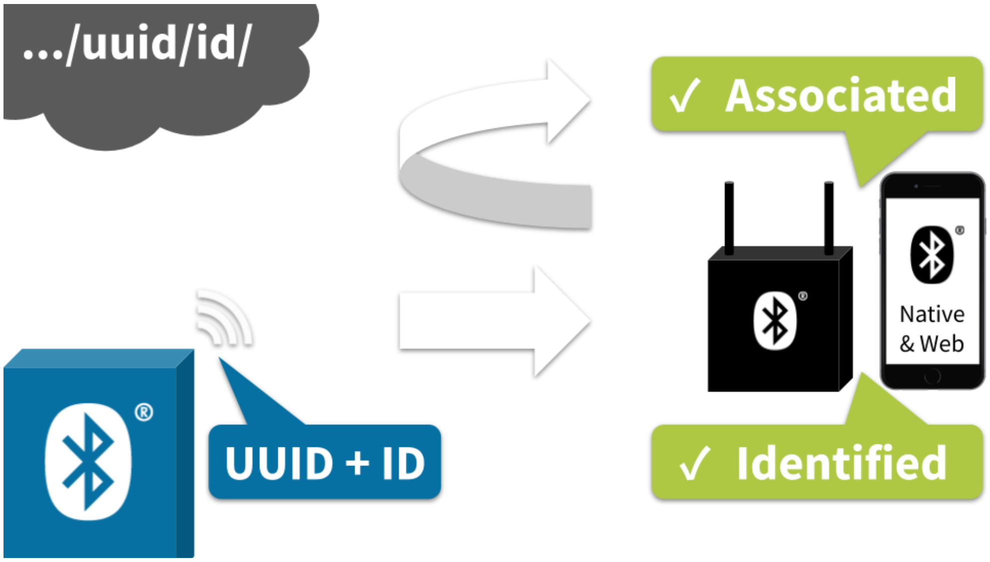
Best practices for BLE identifiers

Assignment of Bluetooth Low Energy (BLE) identifiers for interoperability and interpretability



The TL;DR (Too Long; Didn't Read)

Learn how we at reelyActive assign BLE identifiers based on best practices we've established.

Established how?	When we founded reelyActive in 2012, we recognised that BLE would become the de facto global standard for active RFID, and have been consistently outspoken about best practices.
Why best practices?	BLE devices are all around us, representing people, products and places. Best practices foster spontaneous discovery and interpretation, and maximise interoperability.
Are these observed?	Not especially. As of 2020, we've seen few standard best practices emerge, hence our motivation to create this tutorial and *The InteroperaBLE Identifier!*

Best practices breakdown

This tutorial is organised into four parts as follows:

Links to related tutorials are provided at the end.

Anatomy of a BLE advertising packet [Part 1 of 4]

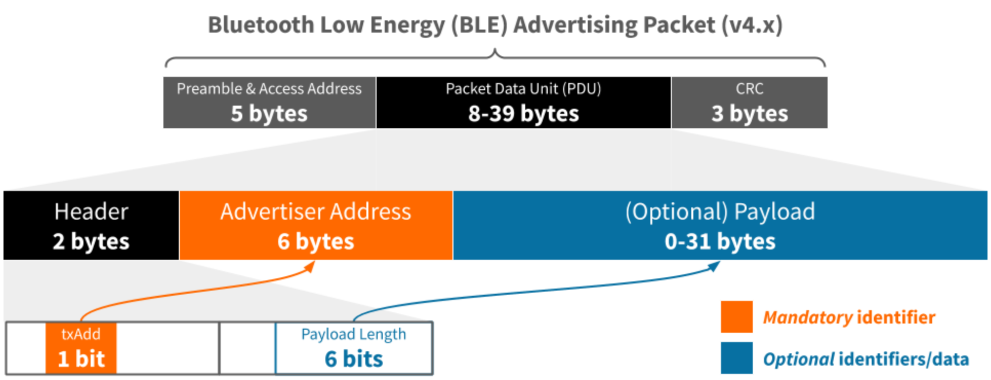
Just enough structure with just enough space for data.

Advertising packet? Here we'll examine only the advertising packets that BLE devices can spontaneously broadcast t any devices listening in range.

Why not other packets? Paired (one-to-one) communication between Bluetooth devices is outside the scope of this tutori but the best practices for identifiers may nonetheless apply.

The advertising packet structure

The diagram below illustrates the structure of a BLE advertising packet with the **Packet Data Unit (PDU)** expanded and identifier content highlighted. The preamble, access address and CRC are typically generated/processed automatically, and are irrelevant to this discussion.



The *mandatory* Advertiser Address

Every BLE advertising packet includes a **48-bit Advertiser Address** and a **single-bit txAdd flag**. The latter indicates whether this address is:

0 I Public	*Unique* identifier from a block of addresses assigned by the IEEE to the device manufacturer
1 I Random	*Random identifier* selected automatically in software, or specifically by the user

⚠ A random identifier *may not be static*. A device may cycle a random identifier at any time.

⚠ A random identifier *may not be unique*. Collisions are nonetheless unlikely unless assignment is non-random.

Other *optional* identifiers

A BLE advertising packet may include an **optional payload of up to 31 bytes** in which additional identifiers and/or data may reside. The Bluetooth Generic Access Protocol (GAP), documented here, affords many payload options. Below are the most common for identifiers.

UUID Service Data Manufacturer Data

Identifier as a UUID

Bluetooth GAP and the advertising packet payload capacity allow for one or more 16-bit or 32-bit service class UUIDs or a single 128-bit service class UUID. Only a 128-bit UUID can be user-generated, as the shorter UUIDs are assigned by the Bluetooth SIG.

A *service* class UUID is intended to represent a *common service* among a class of devices. For instance, a given model of FitBit will advertise a common *service* UUID.

✗ A service class UUID is therefore *not* suited for the *unique* identification of a device.

Platform and OS constraints Part 2 of 4

Just because it's possible doesn't mean it's permitted by the platform or operating system.

Not permitted? Resource constraints, design decisions, privacy concerns and even business models may restric what can be sent, received or accessed.

What's the impact? You might not be able to do *what* you want, or *how* you want to do it, especially if it needs to worl *all* platforms.

The BLE processing pipeline

The diagram below illustrates the processing pipeline between a raw packet over-the-air and the interface with application software under a developer's control. Although the advertising packet structure is *standard*, as presented in the previous section, **the processed data available for a software application is *not standard*:** it depends on the platform and OS.

Similarly, application software is constrained by the platform and OS with regard to any advertising packets it wishes to transmit.

Key constraints by platform/OS

The table below highlights the key constraints with respect to receiving and transmitting identifier data in BLE advertising packets.

	Raw			Espruino	Web
Receive functionality					
Access advertiser address	✓	✗	✓	✓	✗
Access PDU elements	✓	✓	✓	✓	✓
Access raw PDU	✓	✗	✗	✗	✗

Transmit functionality

Specify advertiser address	✓	↻	↻	✓	−
Transmit iBeacon	✓	✓	✓	✓	−
Transmit user-defined payload	✓	✗	✓	✓	−

↻ Mobile devices use a random advertiser address which they cycle regularly (ex: every 15 minutes)

ⓘ reelyActive infrastructure operates raw, without a Bluetooth stack

Identifier interoperability summary

The objective of BLE identifier best practices being to *maximise interoperability*, the platform and OS constraints presented above lead to the following observations:

✗ **Advertiser Address**	Although *mandatory*, access is not universal and mobile devices cycle the identifier periodically in the interest of user privacy (👍).
✓ **Identifier in payload**	All platforms allow *access* in some form to packet payload elements, and allow some form of user-defined payload *transmission*.

The following section presents existing protocols and standards which include identifiers in the BLE advertising packet payload.

Existing protocols and standards [Part 3 of 4]

Eddystone and iBeacon are among the few widely observed open standards.

Whose standards?	Apple introduced the iBeacon open standard in 2013 and Google followed suit with the Eddyston open standard in 2015.
Et tu, Bluetooth?	The Bluetooth Core Specification is the foundation for higher-level protocols such as the above.

Beacons: a brief history

In 2013 Apple unveiled iBeacon to enhance the location awareness and interactivity of their mobile devices. By strategically placing inexpensive battery-powered beacons throughout a space, mobile devices can estimate their position and/or trigger actions based on proximity.

The week of the iBeacon launch, we published this video demonstrating the potential of the *inverse* use case, where the mobile device itself *transmits* iBeacon. We expanded on this use case in this article featured in GigaOM and evangelised the concept at Bluetooth World 2014.

In 2015 Google released Eddystone, similarly intended for battery-powered beacons to transmit information to the mobile device, with Android supporting this use case. Eddystone extends the proximity beacon concept, adding the potential to transmit a (short) URL, telemetry data and encrypted identifiers.